



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

Отчет по лабораторной работе №5 по курсу «Анализ алгоритмов»

*«Организация асинхронного взаимодействия потоков вычисления на
примере конвейерных вычислений»*

Группа: ИУ7-53Б

Студент:

(Подпись, дата)

Дьяченко А. А.

(Фамилия И. О.)

Преподаватель:

(Подпись, дата)

Строганов Д. В.

(Фамилия И. О.)

Москва, 2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Аналитическая часть	4
1.1 Конвейерная обработка данных	4
1.2 Стандартный алгоритм	4
1.3 Алгоритм Кнута — Морриса — Пратта	4
2 Конструкторская часть	6
2.1 Требования к ПО	6
2.2 Разработка алгоритмов	6
3 Технологическая часть	13
3.1 Средства реализации	13
3.2 Сведения о модулях программы	13
3.3 Реализация алгоритмов	14
4 Исследовательская часть	19
4.1 Технические характеристики	19
4.2 Пример работы программы	19
4.3 Время выполнения реализованных алгоритмов	21
ЗАКЛЮЧЕНИЕ	22
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	23

ВВЕДЕНИЕ

Задача поиска подстроки одна из достаточно распространённых в информатике. Строкой называют последовательность символов (в произвольном порядке) взятых из заданного алфавита. Так например, алфавитом могут быть цифры 0, 1 из которых можно составлять неограниченную по длине цепочку символов, например, 0110100110 или 0 [1].

Практическое значение задачи о точных совпадениях трудно преувеличить. Эта задача возникает в широком спектре приложений: текстовых редакторах, информационно-поисковых системах, электронных библиотеках, каталогах и справочниках и т. д. Алгоритмы поиска подстроки в строке также применяются при поиске заданных образцов в молекулах ДНК [2].

Целью данной лабораторной работы является описание параллельных конвейерных вычислений нахождения подстроки в строке стандартным алгоритмом и алгоритмом Кнута — Морриса — Пратта (КМП).

Для достижения поставленной цели необходимо решить следующие задачи:

- описать структуру конвейерной обработки данных;
- описать алгоритмы обработки данных, которые будут использоваться в текущей лабораторной работе;
- определить средства программной реализации;
- реализовать программу, выполняющую заданную конвейерную обработку;
- сравнить и проанализировать реализации алгоритмов по затраченному времени;
- подготовить отчет по лабораторной работе.

1 Аналитическая часть

1.1 Конвейерная обработка данных

Концепция конвейеризации заключается в сегментировании арифметического устройства на отдельные части, каждая из которых выполняет свою подзадачу [3]. Это позволяет организовать передачу данных от одного этапа к следующему, что увеличивает производительность за счет одновременного выполнения нескольких команд на различных ступенях конвейера.

В результате прохода по первой ленте конвейера формируется заявка, содержащая строку и искомую в ней подстроку. При параллельной реализации заявка помещается в очередь на вторую и третью ленту одновременно. Иначе заявка отправляется на вторую ленту, отвечающую за поиск подстроки в строке стандартным алгоритмом. После, заявка, не поменявшая передаваемые данные, передается на третью ленту.

1.2 Стандартный алгоритм

Одним из самых очевидных и одновременно неэффективных алгоритмов является алгоритм последовательно (прямого) поиска. Суть его заключается в сравнении искомой подстроки с каждым словом в основной строке. Алгоритм не является эффективным, так как максимальное количество сравнений будет равно $O((n-m+1)*m)$, где большинство из них на самом деле лишние. Для небольших строк поиск работает довольно быстро, но если в файлах с большим количеством информации последовательность символов будет искаться очень долго [4].

1.3 Алгоритм Кнута — Морриса — Пратта

Метод, использующий предобработку искомой строки и создающий на ее основе префикс-функцию, содержится в алгоритме Кнута — Морриса — Пратта (КМП). Суть этой функции заключается в нахождении наибольшей подстроки, одновременно находящейся и в начале, и в конце подстроки (как префикс и как суффикс). Смысл префикс-функции заключается в том, что неверные варианты могут быть заранее отброшены, а в начале работы могут рассматриваться некоторые вспомогательные утверждения, где для произвольного слова рассматриваются все его начала, которые по совместительству являются его

концами, и выбираются из них самое длинное. Метод КМП использует следующую идею: если префикс (он же суффикс) строки длиной i длиннее одного символа, то он одновременно и префикс подстроки длиной $i-1$. Время работы всей процедуры линейно и есть $O(m)$, несмотря на то, что в ней присутствует вложенный цикл [5].

2 Конструкторская часть

В этом разделе будут представлены схемы и/или псевдокоды реализуемых алгоритмов.

2.1 Требования к ПО

К программе предъявляется ряд требований:

- искомая подстрока имеет размер S от 6 до 10 символов;
- первые две пары символов искомой подстроки совпадают;
- при генерации исходной строки шанс $7/8$ сгенерировать подстроку из S случайных символов;
- при генерации исходной строки шанс $1/8$ сгенерировать искомую подстроку, конкотенирующую с её первой парой символов, т.е. получить подстроку длины $S + 2$ символов;
- на вход подается строка и искомая в ней подстрока;
- на выходе требуется получить файл с записями индексов вхождения подстроки.

2.2 Разработка алгоритмов

В листинге 1 рассмотрен псевдокод стандартного алгоритма поиска подстроки в строке.

Алгоритм 1 — Стандартный алгоритм

Входные данные: Строка *text*, подстрока *pattern*

Выходные данные: Индекс первого вхождения *pattern* в *text*, или
-1 если не найден

```
1  $n \leftarrow \text{length}(\text{text});$ 
2  $m \leftarrow \text{length}(\text{pattern});$ 
3 цикл  $i \leftarrow 1$  от  $n - m + 1$  выполнять
4    $j \leftarrow 1;$ 
5   до тех пор, пока  $j \leq m$  and  $\text{text}[i + j - 1] = \text{pattern}[j]$ 
6     выполнять
7      $j \leftarrow j + 1;$ 
8   конец
9   если  $j > m$  тогда
10    возвратить  $i;$ 
11 конец
12 возвратить -1;
```

Алгоритм Кнута-Морриса-Пратта поиска подстроки в строке использует префиксную функцию. Префиксная функция, вычисляемая для искомой подстроки, содержит сведения о том, в какой мере образец совпадает сам с собой после сдвигов. Эта информация помогает избежать лишних проверок при поиске [2].

В листинге 2 рассмотрен псевдокод алгоритма расчета префиксного массива.

Алгоритм 2 — Расчет префиксного массива	
Входные данные: Подстрока x	
Выходные данные: Префиксный массив π	
1	$m \leftarrow \text{length}(x);$
2	$\pi[0] \leftarrow 0;$
3	$k \leftarrow 0;$
4	цикл $q \leftarrow 1$ от m выполнять
5	до тех пор, пока $k > 0$ и $x[k + 1] \neq x[q]$ выполнять
6	$k \leftarrow \pi[k];$
7	конец
8	если $x[k + 1] = x[q]$ тогда
9	$k \leftarrow k + 1;$
10	конец
11	$\pi[q] \leftarrow k;$
12	конец
13	возвратить $\pi;$

В листинге 3 рассмотрен псевдокод алгоритма Кнута – Морриса – Пратта.

Алгоритм 3 — Кнута – Морриса – Пратта

Входные данные: Строка $w = a_1 \dots a_n$, подстрока $x = b_1 \dots b_m$,
префиксный массив π

```
1  $i \leftarrow 0$ ;  
2  $j \leftarrow 0$ ;  
3  $n \leftarrow \text{length}(w)$ ;  
4  $m \leftarrow \text{length}(x)$ ;  
5 до тех пор, пока  $i < n$  выполнять  
6     если  $a_i = b_j$  тогда  
7          $i \leftarrow i + 1$ ;  
8          $j \leftarrow j + 1$ ;  
9         если  $j = m$  тогда  
10            возвратить  $i - n$ ;  
11         конец  
12     конец  
13     иначе  
14         если  $j = 0$  тогда  
15              $i \leftarrow i + 1$ ;  
16         конец  
17         иначе  
18              $j \leftarrow \pi[j - 1]$ ;  
19         конец  
20     конец  
21 конец  
22 возвратить  $-1$ ;
```

На рисунке 1 изображен последовательный случай реализации конвейера.

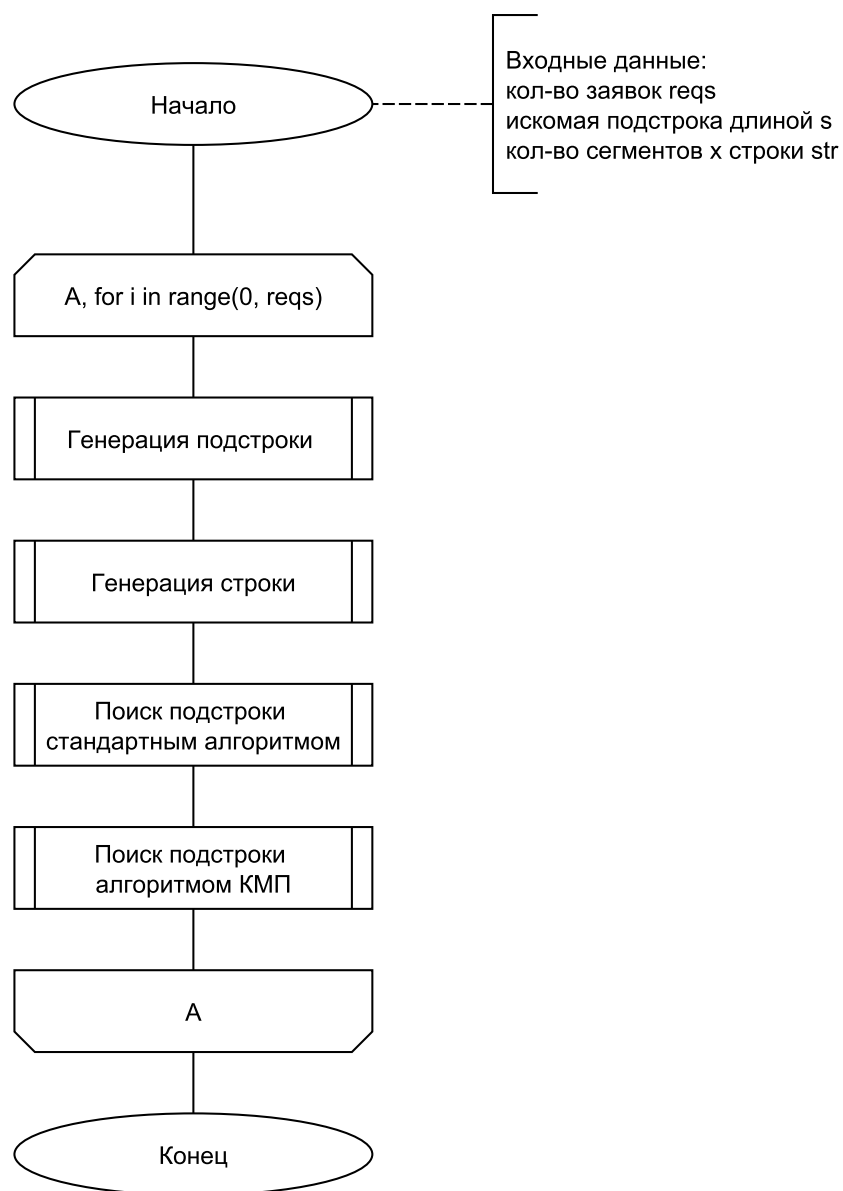


Рисунок 1

На рисунке 2 изображен параллельный случай реализации конвейера. Он заключается в том, что все вторая и третья ленты выполняются одновременно в разных потоках.

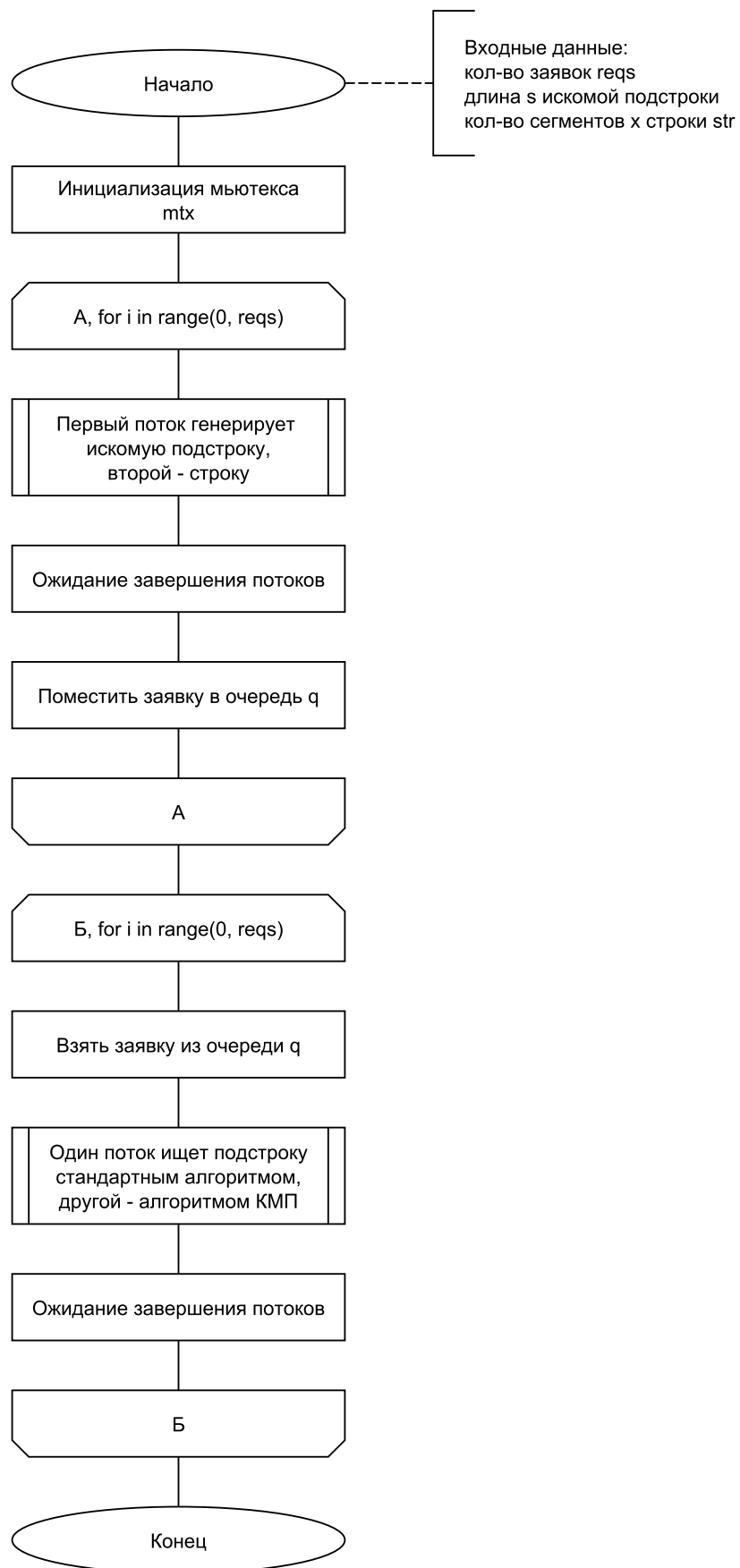


Рисунок 2

При этом, функция записи лога на рисунке 3 содержит критическую секцию, доступ к которой ограничивается мьютексом, который передается каждому потоку.

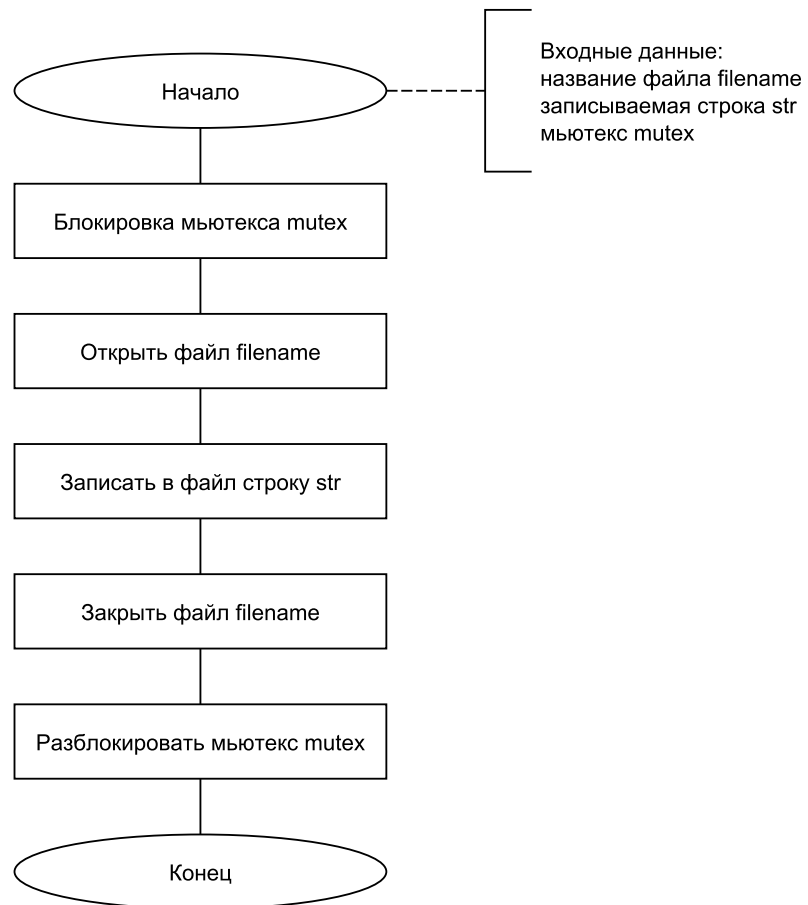


Рисунок 3

3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран C++. Данный выбор обусловлен наличием функционала для работы с потоками `thread` и библиотекой `chrono` для замера времени выполнения алгоритмов.

3.2 Сведения о модулях программы

Программа состоит из следующих программных модулей:

- `main.cpp` — файл, содержащий меню и точку входа в программу;
- `time.cpp` — модуль замера времени выполнения алгоритмов;
- `prefix.cpp` — модуль расчёта массива префиксов;
- `KMP.cpp` — модуль реализации алгоритма КМП;
- `standart.cpp` — модуль реализации стандартного алгоритма;
- `generate.cpp` — модуль генерации данных;
- `conveyor.cpp` — модуль реализации конвейерной обработки;
- соответствующие им файлы заголовков;
- `plot.py` — модуль создания графического представления значений.

3.3 Реализация алгоритмов

В листинге 1 приведена реализация стандартного алгоритма поиска подстроки в строке.

Листинг 1 – Стандартный алгоритм

```
1 void standart(string str, string sub, mutex &mtx) {
2     size_t n = str.length();
3     size_t s = sub.length();
4
5     for (int i = 0; i <= n - s; i++) {
6         int j;
7         for (j = 0; j < s; j++) {
8             if (str[i + j] != sub[j])
9                 break;
10        }
11        if (j == s) {
12            // cout << "Substring found at position " << i << endl;
13            write_in_file("log.txt", "Стандартный: " + sub + "
14                найденавпозиции " + to_string(i), mtx);
15        }
16    }
```

В листинге 2 приведена реализация алгоритма расчета префиксного массива, необходимого для алгоритма КМП.

Листинг 2 – Расчет префиксного массива

```
1 vector<int> get_prefix(string sub) {
2     int n = sub.length();
3     vector<int> prefix(n, 0);
4     int j = 0;
5
6     for (int i = 1; i < n; i++) {
7         while (j > 0 && sub[i] != sub[j])
8             j = prefix[j - 1];
9         if (sub[i] == sub[j])
10            j++;
11        prefix[i] = j;
12    }
13    return prefix;
14 }
```

В листинге 3 приведена реализация алгоритма КМП поиска подстроки в строке.

Листинг 3 – Алгоритм КМП

```
1 void KMP(string str, string sub, mutex& mtx) {
2     vector<int> prefix = get_prefix(sub);
3
4     size_t n = str.length();
5     size_t s = sub.length();
6     int j = 0;
7
8     for (int i = 0; i < n; i++) {
9         while (j > 0 && str[i] != sub[j]) {
10             j = prefix[j - 1];
11         }
12         if (str[i] == sub[j]) {
13             j++;
14         }
15         if (j == s) {
16             // cout << "Substring found at position " << i - s + 1
17                 << endl;
18
19             write_in_file("log.txt", "КМП: " + sub + "
20                 найденaвпозиции " + to_string(i - s + 1), mtx);
21
22             j = prefix[j - 1];
23         }
24     }
25 }
```

В листинге 4 приведена последовательная реализация конвейера.

Листинг 4 – Последовательный конвейер

```
1 void linear(int reqs, int s, int x) {
2     mutex mtx;
3
4     for (int i = 0; i < reqs; i++) {
5         string sub = generate_substring(s);
6         string str = generate_string(sub, x);
7
8         standart(str, sub, mtx);
9         KMP(str, sub, mtx);
10    }
11
12    return;
13 }
```

В листингах 5 и 6 приведена параллельная реализация конвейера.

Листинг 5 – Параллельный конвейер

```
1 void parallel(int reqs, int s, int x) {
2     vector<thread> threads;
3     queue<pair<string, string>> q;
4     mutex mtx;
5
6     for (int i = 0; i < reqs; i++) {
7         threads.emplace_back([&q, s, x, &mtx]() {
8             string sub = generate_substring(s);
9             string str = generate_string(sub, x);
10            lock_guard<mutex> lock(mtx);
11            q.push(make_pair(str, sub));
12        });
13    }
14
15    for (auto& thread : threads) {
16        thread.join();
17    }
18    threads.clear();
19 }
```


Листинг 6 – Продолжение листинга 5

```
1     for (int i = 0; i < reqs; ++i) {
2         auto result = q.front();
3         q.pop();
4         string str = result.first;
5         string sub = result.second;
6
7         threads.emplace_back([&mtx, str, sub]() {
8             standart(str, sub, ref(mtx));
9         });
10
11        threads.emplace_back([&mtx, str, sub]() {
12            KMP(str, sub, ref(mtx));
13        });
14    }
15
16    int i = 1;
17    for (auto& thread : threads) {
18        thread.join();
19    }
20 }
```

В листинге 7 приведена реализация алгоритма генерации искомой подстроки.

Листинг 7 – Генерация подстроки

```
1 string generate_substring(int s) {
2
3     string sub = "";
4
5     for (int j = 0; j < s; j++)
6         sub += 97 + rand() % 25;
7
8     string first_two = sub.substr(0, 2);
9     sub = first_two + sub;
10
11    return sub;
12 }
```

В листинге 8 приведена реализация алгоритма генерации строки.

Листинг 8 – Генерация строки

```
1 string generate_string(string sub, int x) {
2     srand(time(0));
3     string str = "";
4     int s = sub.length();
5
6     for (int i = 0; i < x; i++) {
7         int n = rand() % 8;
8
9         if (n == 7) {
10            str += sub[0] + sub[1];
11            for (int j = 0; j < s; j++)
12                str += sub[j];
13        }
14        else {
15            for (int j = 0; j < s; j++)
16                str += 97 + rand() % 25;
17        }
18    }
19
20    return str;
21 }
```

4 Исследовательская часть

4.1 Технические характеристики

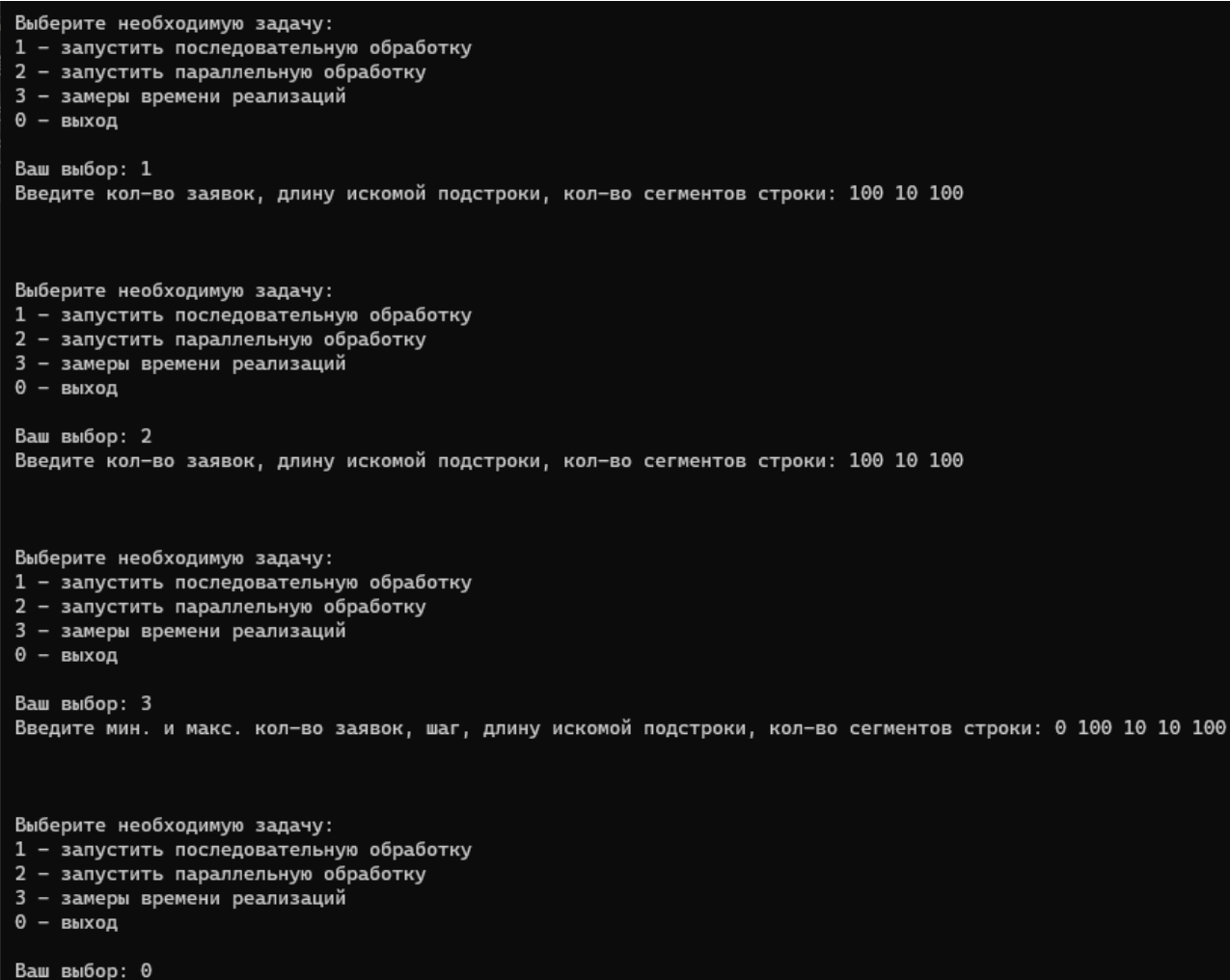
Технические характеристики устройства, на котором выполнялся замерный эксперимент:

- операционная система Windows 11;
- память 16 ГБ;
- процессор 3,6 ГГц 6-ядерный процессор AMD Ryzen 5000 series 5.

Замеры проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только интегрированной средой разработки и непосредственно выполняемой программой.

4.2 Пример работы программы

На рисунке 4 представлен пример работы программы.



```
Выберите необходимую задачу:
1 - запустить последовательную обработку
2 - запустить параллельную обработку
3 - замеры времени реализаций
0 - выход

Ваш выбор: 1
Введите кол-во заявок, длину искомой подстроки, кол-во сегментов строки: 100 10 100

Выберите необходимую задачу:
1 - запустить последовательную обработку
2 - запустить параллельную обработку
3 - замеры времени реализаций
0 - выход

Ваш выбор: 2
Введите кол-во заявок, длину искомой подстроки, кол-во сегментов строки: 100 10 100

Выберите необходимую задачу:
1 - запустить последовательную обработку
2 - запустить параллельную обработку
3 - замеры времени реализаций
0 - выход

Ваш выбор: 3
Введите мин. и макс. кол-во заявок, шаг, длину искомой подстроки, кол-во сегментов строки: 0 100 10 10 100

Выберите необходимую задачу:
1 - запустить последовательную обработку
2 - запустить параллельную обработку
3 - замеры времени реализаций
0 - выход

Ваш выбор: 0
```

Рисунок 4 – Пример работы программы

ПО не выводит сообщения в консоль, но пишет их в лог. На рисунке 5 приведен его пример.

```
1 Стандартный: qrqrjatydimо найдена в позиции 37
2 Стандартный: qrqrjatydimо найдена в позиции 110
3 Стандартный: qrqrjatydimо найдена в позиции 37
4 КМП: qrqrjatydimо найдена в позиции 37
5 Стандартный: qrqrjatydimо найдена в позиции 37
6 КМП: qrqrjatydimо найдена в позиции 37
7 КМП: qrqrjatydimо найдена в позиции 37
8 Стандартный: qrqrjatydimо найдена в позиции 37
9 КМП: qrqrjatydimо найдена в позиции 37
10 Стандартный: qrqrjatydimо найдена в позиции 37
11 КМП: qrqrjatydimо найдена в позиции 37
12 Стандартный: qrqrjatydimо найдена в позиции 37
13 КМП: qrqrjatydimо найдена в позиции 37
14 Стандартный: qrqrjatydimо найдена в позиции 37
15 КМП: qrqrjatydimо найдена в позиции 37
16 КМП: qrqrjatydimо найдена в позиции 110
17 Стандартный: qrqrjatydimо найдена в позиции 147
18 КМП: qrqrjatydimо найдена в позиции 37
19 Стандартный: qrqrjatydimо найдена в позиции 37
20 КМП: qrqrjatydimо найдена в позиции 37
21 Стандартный: qrqrjatydimо найдена в позиции 37
22 КМП: qrqrjatydimо найдена в позиции 37
23 Стандартный: qrqrjatydimо найдена в позиции 37
24 КМП: qrqrjatydimо найдена в позиции 37
25 Стандартный: qrqrjatydimо найдена в позиции 37
26 КМП: qrqrjatydimо найдена в позиции 37
27 Стандартный: qrqrjatydimо найдена в позиции 110
28 Стандартный: qrqrjatydimо найдена в позиции 37
29 КМП: qrqrjatydimо найдена в позиции 37
30 Стандартный: qrqrjatydimо найдена в позиции 37
```

Рисунок 5 – Пример лога программы

4.3 Время выполнения реализованных алгоритмов

Замер времени проводился с помощью библиотеки *chrono* [6], в частности — с помощью структуры *steady_clock* [7].

Результаты измерений времени работы алгоритмов приведены на рисунке 6.

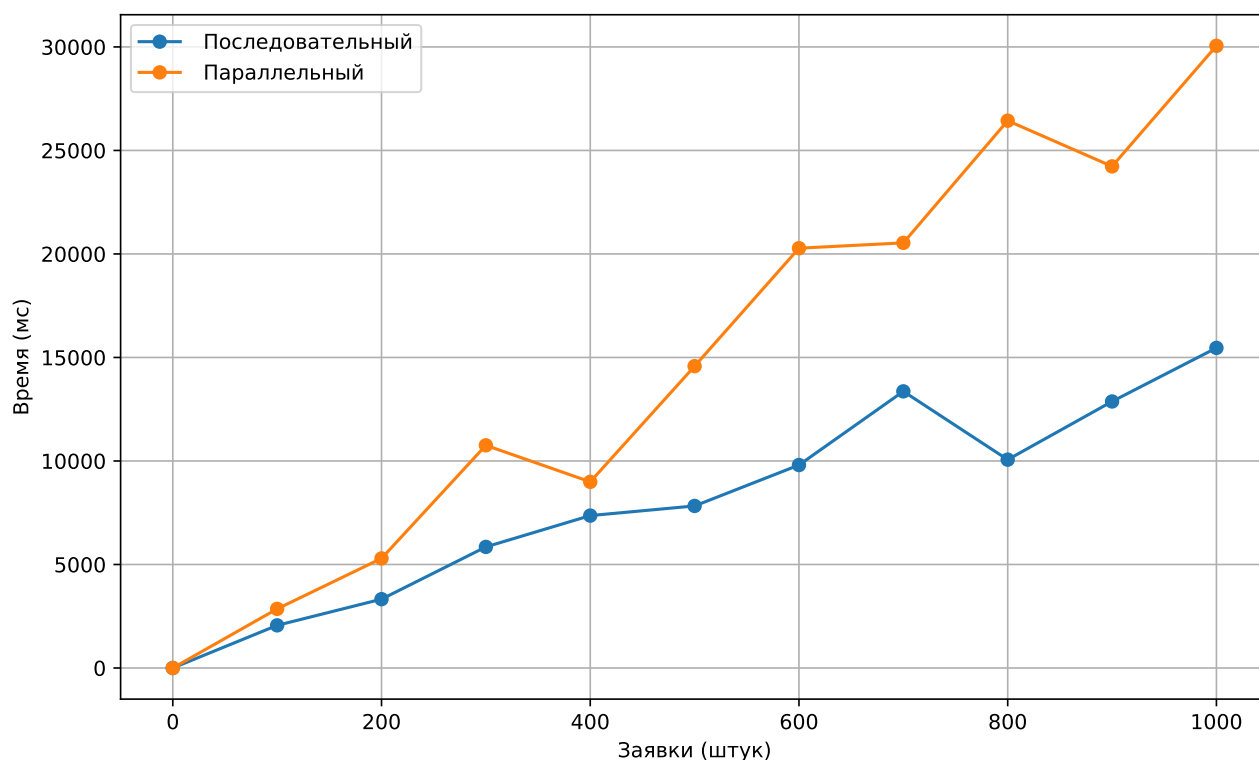


Рисунок 6 – Время выполнения алгоритмов

Время работы параллельной реализации конвейера во всех случаях больше его последовательной вариации.

Вывод

Во всех случаях последовательная реализация конвейера выигрывает по времени у параллельной. Происходит это из-за того, что время, которое тратится на управление потоками, не перекрывает «выигрыш» во времени, который достигается за счет одновременной обработки данных. Как следствие — последовательная реализация конвейера работает быстрее.

ЗАКЛЮЧЕНИЕ

Цель работы достигнута: описаны последовательные и параллельные конвейерные вычисления нахождения подстроки в строке стандартным алгоритмом и алгоритмом Кнута — Морриса — Пратта (КМП), проведено сравнение времени работы.

В ходе выполнения лабораторной работы были решены все задачи:

- описана структура конвейерной обработки данных;
- описаны и реализованы используемые алгоритмы;
- определены средства программной реализации;
- реализована программа, выполняющая заданную конвейерную обработку;
- проведено сравнение реализаций алгоритмов по затраченному времени;
- подготовлен отчет по лабораторной работе.

В результате исследования выяснилось, что для задачи поиска подстроки в строке с помощью конвейерной обработки следует использовать последовательную реализацию, поскольку она быстрее параллельной при кол-ве заявок до 1000. В среднем выигрыш составляет 40%.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Пелевин М.* Поиск подстроки в строке. — URL: <https://markoutte.me/students/substring>.
2. *Алексеевко А.* Информационная чувствительность алгоритма Кнута-Морриса-Пратта // Задачи системного анализа, управления и обработки информации. — 2010. — С. 5.
3. Архитектуры и топологии многопроцессорных вычислительных систем / А. Богданов [и др.] // М.: Интернет-университет информационных технологий. — 2004.
4. *Вирт Н.* Алгоритмы и структуры данных. Учебное пособие. — ДМК Пресс, 2010.
5. *Солдатова Г. П., Татаринов А. А., Болдырихин Н. В.* Основные алгоритмы поиска подстроки в строке // Academy. — 2018. — 5 (32). — С. 8—10.
6. *Microsoft.* <chrono>. — URL: <https://learn.microsoft.com/ru-ru/cpp/standard-library/chrono?view=msvc-170>.
7. *Microsoft.* Структура steady_clock. — URL: <https://learn.microsoft.com/ru-ru/cpp/standard-library/steady-clock-struct?view=msvc-170>.